IDEAS FOR THE PROVISION OF ONTOLOGY ACCESS IN GRID ENVIRONMENTS

Miguel Esteban Gutiérrez
Ontology Engineering Group
Universidad Politécnica de Madrid
Campus de Montegancedo s/n, 28660, Boadilla del Monte
Madrid, Spain
mesteban@fi.upm.es

Asunción Gómez-Pérez
Ontology Engineering Group
Universidad Politécnica de Madrid
Campus de Montegancedo s/n, 28660, Boadilla del Monte
Madrid, Spain
asun@fi.upm.es

Abstract Ontologies are the backbone of the Semantic Web. Current grid architectures do not consider their usage, and there are no protocols nor standards in the Grid community for dealing with them. Therefore, the provision of appropriate means for accessing, querying and using ontologies effectively is a key factor if we want to enrich the current grid with semantic technologies and to support progress towards the next generation Grid, that is, the Semantic Grid.

Keywords: Ontologies, semantic grid, semantic technologies, WS-DAIOnt, WS-DAI, OGSA

1. Introduction

The increasing use of semantic technologies has reached almost every computer-related field, including the Grid. The next generation Grid should virtualise the notion of distribution in computation, storage, and communication over unlimited resources using well-defined computational semantics, as the Semantic Grid [7] is proposing. A grid node may provide new resources and services and their functional and non functional properties should be explicitly defined by means of *ontologies*, formal and explicit specifications of shared conceptualizations [15]. Therefore, if semantic technologies are to be used, it is fundamental to provide the appropriate means for accessing, querying and using ontologies in the Grid.

In this chapter, we analyse the problem of accessing, querying and using ontologies concerned with the current Grid architecture, taking as starting point the lessons learnt about this topic in the Semantic Web.

The chapter is organised as follows: Section 2 collects some of the most important lessons learnt in the Semantic Web regarding ontology access, query and use. Section 3 comprises an analysis of the ontology access problem in the context of the current Grid. Section 4 presents WS-DAIOnt, a proposed mechanism for ontology access in the current Grid. Finally, Section 5 concludes with the current state of development.

2. Lessons Learnt from the Semantic Web

Recently, the W3C has recommended three languages: RDF [1], RDFS [2] and OWL[3], to represent knowledge in the Semantic Web.

In addition, several ontology development tools (i.e., Protégé[4], WebODE[5], KAON[6]) support the creation of ontologies in such languages. There are also ontology query languages like SPARQL [13], RDQL [14], RQL [12], SeRQL [5] used for retrieving RDF(S) and OWL ontologies, and inference engines like FaCT[7] and RACER [11] that infer knowledge and data that are not explicitly declared in the ontologies. Normally, these querying and inference tools are strongly related to the language in which the ontology is implemented. Such languages differ in their expressiveness (the kind of knowledge that can be represented) and in their inference mechanisms (the kind of reasoning they

[1] http://www.w3.org/RDF/
[2] http://www.w3.org/TR/rdf-schema/
[3] http://www.w3.org/2001/sw/WebOnt/
[4] http://protege.stanford.edu/
[5] http://webode.dia.fi.upm.es/WebODEWeb/
[6] http://sourceforge.net/projects/kaon/
[7] http://www.cs.man.ac.uk/ horrocks/FaCT/

carry out). For a detailed description and comparison of languages and tools we recommend [10].

The diversity of existing ontology languages and tools causes the *translation problem*, which appears when an ontologist decides to reuse an ontology (or part of it) with a tool other than the one used in its development, or in a language other than those in which the ontology is available. On the other hand, several APIs and query languages permit accessing ontologies implemented in a given language and an ontology user (or an application that uses the ontology) should know how to retrieve the ontology content using those APIs. As example, we can say that RDF(S) ontologies can be stored in Sesame[8], 3store[9], Joseki[10], Jena[11], Kowari[12] or even Oracle with its support for RDF(S)[13], and each has its own means for accessing the RDF(S) ontologies.

In this scenario interoperability and portability problems arise since the heterogeneity (different characteristics, properties and capabilities) of the languages and tools used for the development and storage of the ontologies might prevent the reutilization of these ontologies in different infrastructures because of their technological differences, namely, their limitations, restrictions and requirements.

At present, there are some language specific initiatives in the Semantic Web community devoted to solving specific problems, such as the W3C SPARQL query language, created to provide a RDF(S) query language for accessing to RDF(S) stores[14] [13], or the DIG interface, targeted to providing a common API for description logic-based systems interoperability [4].

Despite all these initiatives, the Semantic Web community does not have a standard mechanism or protocol for accessing ontologies implemented regardless of the language and tool used for its development.

3. Possibilities for Providing Ontology Access in the Grid

Up to now, current grid architectures have not taken into consideration ontology use; therefore, no protocols nor standards are available in the grid community to access and use them.

To use ontologies as other resources in the Grid, we must be able to access their contents physically, as we do with any other available resource. Therefore, the first requirement for using ontologies is to have the appropriate means for

[8] http://www.openrdf.org/
[9] http://threestore.sourceforge.net/
[10] http://www.joseki.org/
[11] http://jena.sourceforge.net/
[12] http://www.kowari.org/
[13] http://www.oracle.com/technology/tech/semantic_technologies/
[14] Targeted at retrieving data, not creating, deleting or updating data.

accessing them. Building on these basic capabilities, it will be possible to develop and deploy ontology-based functionalities in the Grid.

In this section we discuss where ontology access fits in the *Open Grid Services Architecture* (OGSA), and how ontology access services can be implemented. By *ontology access* we mean the mechanism or protocol needed for providing physical access to ontologies; by *ontology access services* we refer to the set of services that provide the means for accessing ontologies that are deployed as resources inside an OGSA-based grid.

3.1 Laying Ontology Access Services in OGSA

The Open Grid Services Architecture specification [8] is the blueprint for standard-based service-oriented grid computing. The specification collates the requirements for such an architecture[15] and also identifies the set of capabilities (offered as services) that may be needed in order to satisfy the defined requirements: infrastructure services, job management services, data services, resource management services, security services, self-management services and information services. For a detailed description of both requirements and capabilities, please refer to [8].

Ontologies can be queried as to their content. Content includes the concepts and relationships, as well as intensional information about those concepts, as for example, the definitions that apply to a particular class. Ontology access services should then provide access to all this information and even support queries over this information. Thus, ontology access services can be seen as a particular type of data service, a service that holds some data and provides mechanisms for creating, retrieving, updating and deleting these data. According to this, the most sensible mechanism for providing ontology access services should be based on the existing infrastructure in OGSA for data access.

The following subsection reviews the data access and integration facilities in OGSA, as these must be known to fully understand the rest of the chapter.

3.1.1 The OGSA Data Access and Integration Facilities.

The *Global Grid Forum (GGF) Data Access and Integration Working Group* has a number of specifications that support data access on the Grid. The *Web Services Data Access and Integration Core Specification* [1], a.k.a. WS-DAI, and the accompanying realizations [2, 6, 3], provide a general mechanism for defining data services (whose key characteristic is that it offers the possibility of updating and retrieving the data from the data resource with which it interfaces) and specialised mechanisms for accessing specific data resources respectively.

[15] They range from interoperability and support for dynamic and heterogeneous environments and resources to resource sharing across organizations to data access to scalability, availability and extensibility.

The top level WS-DAI specification provides a basic and extensible framework for defining data service interfaces, messages and properties. With such a framework, the set of port types, operations and properties – which are needed to provide access to specific data resources – can be defined in a standard way. However, WS-DAI does not describe the particular interactions it performs with the data resource (in terms of, for example, query languages).

The underlying WS-DAI realizations describe the specific operations needed to interact with specific data resources, i.e., relational databases using SQL, XML sources using XQuery, etc. The upper level specification is irrelevant to the types of query that get passed through — they say nothing about the query language, result formats etc. The interfaces, messages and properties used must be defined in accordance with the WS-DAI framework.

Note that when building an application that interacts with a WS-DAI resource, it should be known beforehand which kind of resource will be used so that the appropriate WS-DAI realization is used. WS-DAI provides the most general properties of the data service as a grid service that the application may need to know about, while the realization provides the specific mechanisms for interacting with the data service.

3.2 Ontology Access through the OGSA Data Access and Integration Facilities

In order to integrate the ontology access services within the WS-DAI framework described in the previous section, several alternative approaches can be adopted according to how the framework is used and where the new services fit.

The approaches range from the mere use of the WS-DAI framework to the extension of this framework to fit our purposes. Here, we also present the idea of *abstract realization*, a realization which is not a plain realization of WS-DAI but a set of guidelines that explain how to use the WS-DAI framework for defining sets of related realizations.

In this section, we analyse each of the possible approaches and give examples that showe possible implementations of each approach; the Semantic Web languages RDF(S) and OWL illustrate these examples.

3.2.1 Several vanilla realizations. The first and most naïve approach consists in providing a specific access mechanism for each ontology language that is to be used; in our case, this means supplying a WS-DAI realization to each possible Semantic Web ontology language.

The specific realizations must adhere to the syntax guidelines given by the WS-DAI framework for defining the access mechanism. The concrete technical aspects needed to access data sources containing ontologies developed with each

specific Semantic Web ontology language are defined in its related realization. However, these realizations may have nothing in common, as the semantics of each realization is conditioned by the necessities and requirements of the access mechanism provided for each language, as it happens in the following example.

Here we have two basic realization designs for accessing respirces, one for RDF(S) and other OWL, both developed independently.

The RDF(S) realization may provide access to RDF(S) by relaying on the graph nature of the RDF model: every model is composed of a set of nodes and links between the nodes. The nodes represent specific resources while the links represent properties of the source node resources.

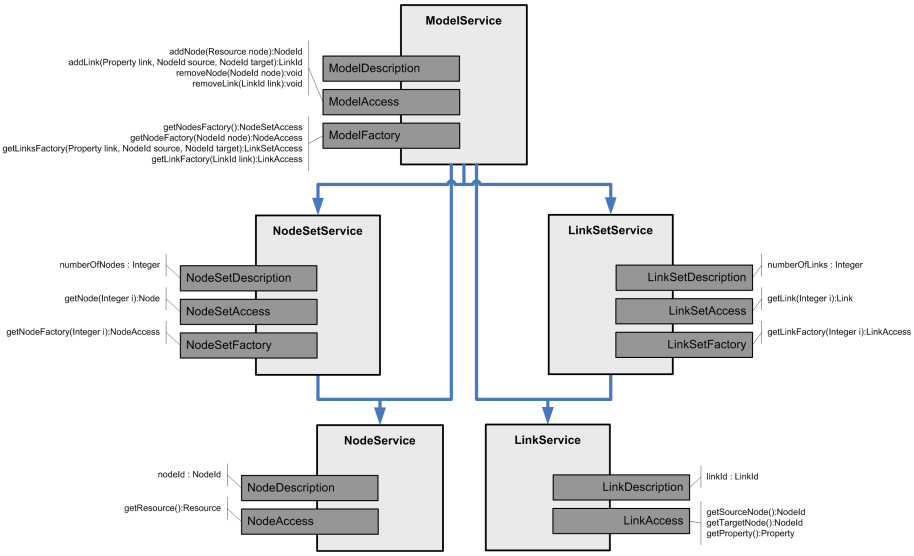

Figure 1. Sample RDF(S) vanilla realization

Following this idea, the realization can provide a set of interfaces that deliver functionalities for dealing with models with data structures that represent nodes (resources) and links (properties). Figure 1 represents a sample set of interfaces (already grouped in services) and the signatures of some of the messages provided by the main interfaces.

The design of the OWL realization can follow other approach, as for instance, an object-oriented one. According to this approach, the realization would provide interfaces which deal with data structures that mimic the conceptual elements defined in the OWL model: classes, properties, individuals, restrictions, etc. Figure 2 represents a sample set of interfaces and the signatures of some of the messages provided by the main interfaces.

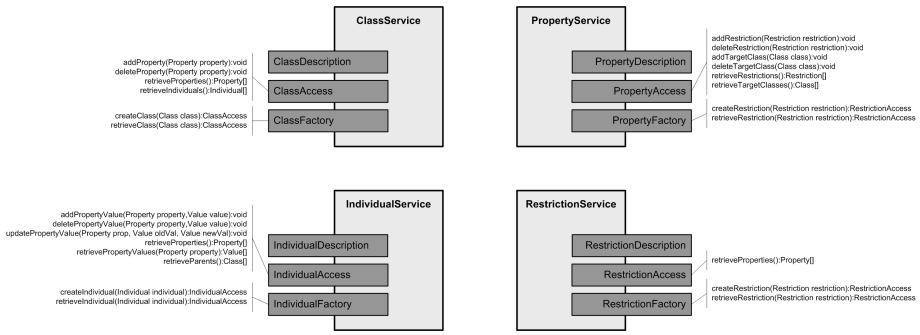

Figure 2. Sample OWL vanilla realization

Even though both realizations offer valid mechanisms for accessing their respective resources, we cannot ensure, according to this design approach, any homogeneity degree between the realizations, as there are no common guidelines that guarantee such homogeneity:

- *Data model structure:* nothing is said about how to structure the data model that is to be used in the messages. In this scenario, the RDF(S) realization uses a graph-based data model, while the OWL realization employs a component-like data model. Therefore, the interaction with the ontology services needs to be adapted to the specific operational model derived from the data model.

- *Naming conventions:* each realization uses its own prefixes and suffixes for denoting the names of the messages; for instance, the RDF(S) realization defines the messages adding the prefixes ‘get’, ‘add’ and ‘remove’ whilst the OWL realization names the messages with similar semantics using the prefixes ‘retrieve’, ‘create’ and ‘delete’; furthermore, this latter realization defines messages with different semantics using a prefix (add) that clashes with other in the RDF(S) realization. When switching from one realization to other, the user will have to use another data model and to learn which are the concrete semantics of each type of message (being the type defined by the prefix and suffix combination).

- *Access modalities:* the RDF(S) realization provides messages for creating, retrieving and deleting contents, while the OWL realization has an additional functionality for updating the contents. Therefore, if we switch from an ontology developed in OWL to another developed in RDF(S), we have to simulate this extra functionality, as it is not present in the RDF(S) realization. It might happen that missing functionalities cannot be simulated due to the specific design of the other realization; in that

case the patterns of interaction with the ontology in the user's application should have to be reviewed.

- *Access granularity:* in the case of the RDF(S) realization, access to the contents is provided in different granularity degrees: we can interact directly with a model, with a set of components (nodes and links) or with specific components. The richer the messages provided in each level are, the better the interaction with the RDF(S) resource will be. On the other hand, the OWL realization just provides access to specific components of the model. Again, when switching from one realization to other, we will have to reorganize the ontology-based business logic in order to use the services properly.

- *Architectural organization:* messages with similar semantics are defined in different conceptual components. Whilst the creation of components in the RDF(S) realization is carried out in the ModelAccess interface, in the OWL realization this is delegated to the Factory interfaces associated to each component.

According to this scenario, the same interoperability problems that arise in the Semantic Web community appear when following this design approach: we end up having different mechanisms for accessing ontologies represented in RDF(S) and OWL, each one designed according to different criteria, which provide zero interoperability. Therefore, the ontology access service client must know beforehand the language in which the ontology is available, because he/she will have to use one or other realization for modelling the ontology-based business logic of the application. As it follows, switching from one language to other might cause severe changes in the ontology-based business logic, which is the main issue we are trying to solve.

3.2.2 Two-layer realization. The idea here is to separate the common operations from the specific ones, following a two-layer organisation. On the one hand, the upper layer would contain a base WS-DAI realization defining the common specific operations that must be provided by every ontology access mechanism. On the other hand, the lower layer would contain WS-DAI realizations based on the base realization; each realization, which should be related to an ontology language, would define the specific operations for accessing ontologies developed in that particular ontology language.

In terms of operations, the two-layer realization approach introduces the idea of a common *API* that must be followed by each final realization (and so by each implementation). There are several ways of creating this API, here we present two of them:

- *Functional approach.* The API contains the operations that represent functionalities for ontology access and management similar to those offered by ontology resources. According to this approach, the definition of the operations would be driven by the possibilities of the ontology resources.

  Functionalities can be selected in various ways, for instance, according to the desired granularity of the operations (the finer the grain detail is, the more functionalites will have to be provided by the API) or according to the size of the target API: we may want to minimize the number of functionalities provided by the API, so this becomes simple although rigid; or to maximize the number of operations so it becomes more flexible but rather complex.

- *Conceptual approach.* The API contains operations that deal with the conceptual elements available in the knowledge representation formalism of the ontology resources. Following this approach, the definition of the operations is driven by the necessities of the conceptual model, nor by the way the ontology resource deals with it, i.e. managing taxonomies, reasoning over inherited properties, etc.

  The number of operations of the API would depend on the modelling elements chosen (the more elements, the more functionalities needed for dealing with them) and the orthogonality of the operations desired (the more independence between the functionalities and the operands, the more operations to provide).

Thanks to this two-layer organisation, the basic ontology access interfaces can be standardised, thus lowering the risks of interoperability issues in the upper layer, that is, the one that contains the common operations. In Figure 3 we can see that the SchemaAccess interface defines common taxonomical management operations such as retrieving the parents and siblings of a given class, and these operations are common to both underlying realizations.

Unfortunately, no standardisation guidelines are provided for defining the specific operations to be set up in the realizations of the lower layer, nor which kind of operations can be defined in those interfaces. Therefore, interoperability problems might appear in some particular features of concrete languages.

We can see in Figure 3 an example of this case. It shows that the new operations added in the RDF(S) realization just provide extra functionalities for dealing with the elements defined in the base realization following the same naming convention, whereas the OWL realization provides new functionalities for operating over specific elements of the OWL model (restrictions) with its own naming conventions.

Regardless of the approach taken for the development of the API, enforcing the fulfillment of an API poses serious disadvantages. On the one hand, if

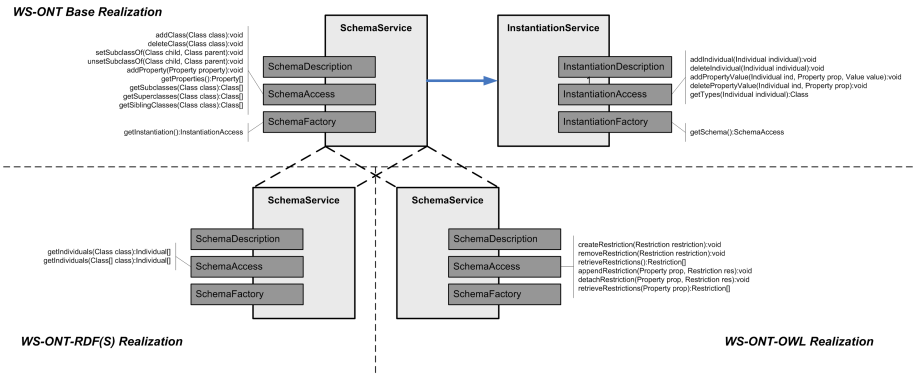

Figure 3. Sample two-layer realizations for RDF(S) and OWL

the API that is to be fitted defines more operations than those provided by the resources, we may have a set of operations that not usable that will weaken the operational model defined by the API. On the other hand, if the API defines fewer operations than those provided by the resources, the potential of those resources is weakened because many of its capabilities will not be usable through the API.

Despite these disadvantages, we gain a strong advantage when APIs are enforced: the standardisation of the operations that will be used hereafter, which helps to reduce interoperability problems across realizations and implementations.

3.2.3 An extension of WS-DAI. Another approach consists in providing an extension to WS-DAI that, using the WS-DAI framework as basis, defines the specific structural elements needed for defining ontology access mechanisms, which are not defined in the basic WS-DAI framework.

Once the extension is defined, it would be used as the basis for creating specific realizations that would provide ad-hoc access mechanisms to access ontologies developed with concrete knowledge representation formalisms or languages. However, this approach has its pros and cons, as we will see in the following example.

In our sample extension, ontology resources (an specific type of data resources) are composed of *components* which, in their turn, may also be composed of other components. For dealing with these, an extra type of interface will be used. The interface will be named appending the suffix ‘Components’ to the name of the data resource it operates with. The interface will provide

BREAD functionalities [16], named with preffix the type of operation and the suffix 'Component'.

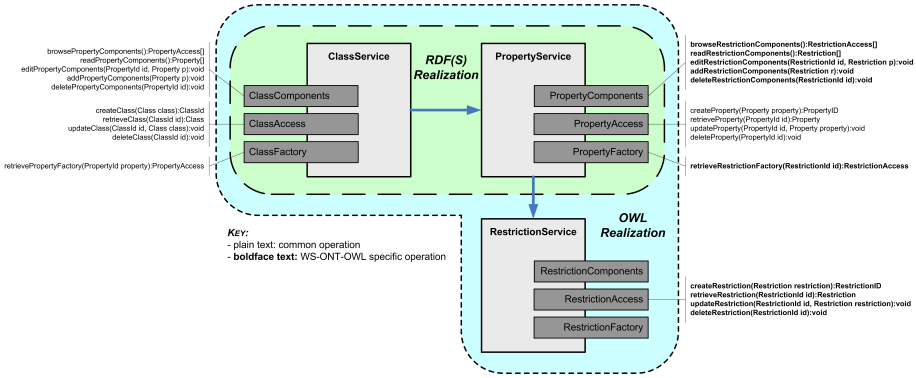

Figure 4. Implementations of the sample extension of WS-DAI

Figure 4 shows implementations of the sample extension for RDF(S) and OWL. Thanks to the extra structural elements defined in the extension, some kind of homogeneity is achieved, i.e. operations share a naming convention, messages with similar semantics are grouped in the same interfaces, etc. Thus, the interoperability problem decreases although unfortunately this effect is limited to these new structural elements.

However, in spite of the homogeneity achieved, the use of extra structural elements makes ontology access services different from plain WS-DAI data access services; therefore, whenever a user needs these services, he/she must know about their specificities to use them properly.

This requirement implies interoperability problems between data services: while a user of ontology access services would be able to understand and exploit plain WS-DAI data services, a user of plain WS-DAI data services would not be able to utilize the ontology access services since he/she has no way of inferring the semantics of the new structural elements used in the declaration of the ontology access services.

3.2.4 An *abstract* realization. This last approach is based on the two previous ones and is targeted at solving the problem of enforcing the fulfillment of a common API, and at creating extra structural elements.

The idea hera is to provide a means for defining the capabilities that might be offered via an ontology access service and the ways in which they are offered,

[16] Browse, read, edit, add and delete.

and to provide a mechanism for publishing the specific capabilities implemented by an ontology access service, so a client can discover and exploit them. This can be done by means of an *abstract* realization.

As we reviewed in section 3.1.1, the objective of the base WS-DAI core specification is to define a base framework for defining data access services that can be adapted to particular necessities. To achieve this, the base specification provides a set of patterns that defines messages and properties. With these patterns, concrete realizations define the set of WS-DAI-based elements needed for accessing a specific kind of data resource.

An *abstract* realization is a realization that does not define specific messages nor properties for providing a particular access mechanism; it defines a set of WS-DAI compliant patterns for defining interfaces, messages and properties oriented to the specification of an adaptable set of related data access mechanisms.

In our case, the abstract realization should be created for defining ontology access mechanisms. Later realizations of this abstract realization can choose which capabilities to implement and then define them with the patterns found in the base abstract realization. Let's see in the following example how this could be achieved.

The first step consists in selecting the elements of the data model that is to be supported. In the case of the RDF(S) and OWL languages, these elements are well-known and defined. Therefore, we could say that the ontology access mechanism must be able to operate over the union of both models: classes, properties, restrictions, individuals, etc. Specific realizations will choose which elements to support. In order to shorten the example, we can think that the valid elements are just classes, properties and restrictions, and that classes and restrictions are linked to properties (and viceversa).

Then, we have to select which kind of operations we want to provide in order to operate over the supported data model. In our example we show basic CRUD operations: create, retrieve, update and delete. The create operation provides an *id* for the element created, and that *id* is used in the rest of the operations for referring to that concrete element.

Once we have defined the data model and the way we can operate with it, we have to define the patterns that will drive the definition of the related infrastructure:

- *Interface creation patterns*:

  PATTERN I1: A description interface named *'element'Description* will be created for each element of the model.

  PATTERN I2: An access interface named *'element'Access* will be created for each element of the model.

PATTERN I3: If an element is linked to any other element, a factory interface named *'element'Factory* will be created for that element.

- *Messages creation patterns*:

  PATTERN M1: A message named *'operation''element'* will be created in the appropriate access interface for each *'element'* that supports the operation *'operation'*[17].

  PATTERN M2: If an element e_i is linked to another element e_j, a message named *retrieve'*e_j*'Factory* will be created in the factory interface of the element e_i.

- *Properties creation patterns*:

  PATTERN P1: A property named *'linkedTo'* will be defined in each description interface. If an element e_i is linked to any other element e_j the value of the property will be the list of elements e_j to which e_i is linked. Otherwise, the property will be nil.

  PATTERN P2: A property named *'operationsSupported'* will be defined in each description interface. If an element e_i supports an operation op_j, the value of the property will be the list of operations op_j supported by the element e_i.

With all of these patterns we can then produce our RDF(S) and OWL realizations (see Figure 5). As we can see, in the case of classes and properties, the names of the interfaces and messages are shared in both RDF(S) and OWL, so switching from one realization to other (when dealing with classes and properties) would not require changes in the ontology-based business logic.

Following this approach, each realization can choose the capabilities required, and by means of the patterns defined in the abstract realization, the interfaces, messages and properties are defined in an homogeneous and standard way.

3.3 Conclusions

The OGSA specification defines a set of requirements that must be fulfilled by any implementation of the specification; it also defines a set of capabilities (services) that might be offered via implementations to fulfil these requirements.

According to these requirements and capabilities, the data access and integration facilities defined by OGSA constitute the most sensible niche for fitting the ontology access services inside the OGSA architecture. The data access

[17] op_j is one of 'create', 'retrieve', 'update', 'delete'

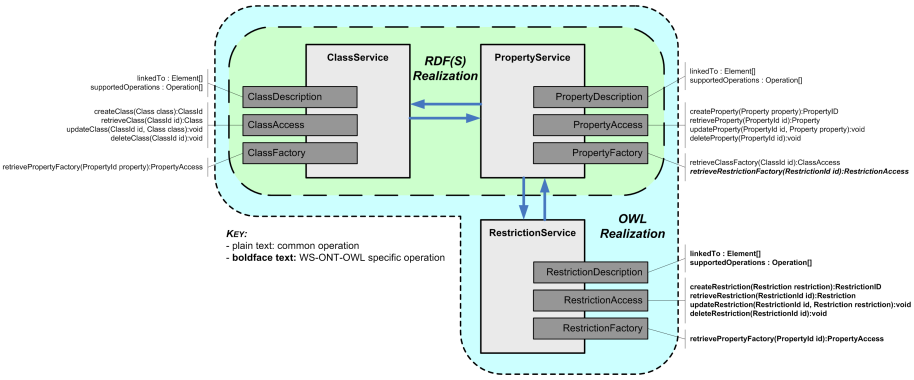

Figure 5. Sample RDF(S) and OWL realizations using the abstract realization approach

and integration facilities are governed by the WS-DAI specification, which is still under development inside of the GGF.

In the previous subsections, we have presented several approaches that provide ontology access using the WS-DAI specification as a basis; these approaches range from the realizations to the extensions.

The first approach clearly requires complete new implementations for any additional language introduced, as nothing is reused from other realizations. The second approach, on the contrary, provides some reuse by means of designing a two-layer mechanism that proposes a set of common functionalities. The cost of such reuse is the issue of factoring out a sensible subset of functionalities valid for any possible ontology language, which may end up posing interoperability problems between the services and the ontology resources.

The third approach consists in creating an extension to WS-DAI. This alternative suggests that there are characteristics and properties which distinguish ontology access services from vanilla data access services. Nevertheless, exploiting these differences by means of extra structural features may imply interoperability problems between plain data services and ontology access services.

Finally, the fourth approach suggests that there might be characteristics that distinguish ontology access services from plain data services, but also, that there are subtle differences between ontology resources that must be also taken into consideration. The fourth approach proposes a way for defining operations in a common format (taking into account these differences between ontology resources), so that some kind of standardisation is introduced in the underlying realizations.

Therefore, and bearing in mind the objectives of reducing the amount of different ontology access mechanisms and of facilitating the interoperability between them, the best design approach is the abstract realization one.

4. WS-DAIOnt: a Proposal of an Ontology Access Mechanism in the Grid

The WS-DAIOnt specification [9], which is the short term for "Web Services Data Access and Integration: The Ontology Realization", and the accompanying realizations (WS-DAIOnt-RDF(S), . . .) define the data access infrastructure needed for dealing with ontologies in grid environments.

WS-DAIOnt is based on the WS-DAI specification and provides a framework for defining ontology access service interfaces by means of the WS-DAI vocabulary, and for enhacing it with the patterns and properties needed to provide specific ontology access mechanisms. Specific ontology data sources are then addressable according to concrete WS-DAIOnt realizations, i.e. WS-DAIOnt-RDF(S).

In the following subsections, both the foundations of WS-DAIOnt and the components of WS-DAIOnt will be described.

4.1 WS-DAIOnt Foundations

The WS-DAIOnt specification is being designed following the abstract realization approach described in Section 3.2.4. The foundational pillars that drive the design of the specification are the following:

- *Unified basic terminology*. Currently, knowledge representation formalisms use their own terminology for naming the knowledge modelling components (ontology elements) they use. Thus, frames-based formalisms use the name 'class' for referring to what it is named 'concept' in description logics.

  Whereas humans are able to match the names as synonyms and use both of them indistinctly, software agents are not able to do so. Therefore, interoperability problems might appear because of this terminology tangle.

  WS-DAIOnt defines a neutral vocabulary for naming the ontology elements to be used when dealing with ontologies in grid environments, taking into account the specific modeling components of different knowledge representation formalisms (frames, semantic networks, description logics . . .)

  This common and standard vocabulary avoids the use of multiple different vocabularies that would hamper the understanding of the provided data components and functionalities.

- *Ontology components relationships patterns*. Each knowledge representation formalism defines a set of modeling elements and the way they are related to each other. For instance, in the frames formalism slots are

defined locally (in frames), whereas in description logics properties are defined globally (and can then be restricted to specific classes).

WS-DAIOnt defines how to specify the concrete ways in which ontology components can be related, and which is the expected semantics of these relationships, so clients can deduce how to conceptually use them properly.

- *Ontology components usage patterns*. WS-DAIOnt defines how the interfaces, messages and properties must be specified in terms of WS-DAI patterns, in order to provide functionalities in a standard way. Therefore, clients can deduce how expected functionalities have to be exploited.

- *Ontology access services behaviours*. WS-DAIOnt defines the expected behaviour of the predefined common components and functionalities, so that every concrete implementation must adhere to these behaviours. Therefore, clients may expect some kind of homogeneous behaviour across realizations and implementations.

4.2 WS-DAIOnt Components

The two main components of WS-DAIOnt are the *WS-DAIOnt Data Model* and the *WS-DAIOnt Port Types*.

The WS-DAIOnt Data Model defines how the data managed by the specified interfaces is virtually structured. The data model works as a metamodel from and to which other knowledge representation formalisms may be mapped — by means of these mappings the interfaces provide a common way for accessing heterogeneous ontologies.

The data model defines the unified terminology to be used in WS-DAIOnt regarding the data components and also defines the possible relationships patterns among them.

The organization of the data model has two dimensions:

- *Layered structure*. The components are divided in two layers, the core layer and the extended layer. On the one hand, the core layer contains the common modeling components found in most of the representative knowledge representation formalisms. On the other hand, the extended layer will contain those modeling elements not considered for the core layer because of their specificity or because they are yet to appear.

- *Model and data separation*. The components are also divided with regard to their concerns: those used for the conceptualization are grouped together in the model part, and those used for dealing with individuals are grouped in the data part.

The WS-DAIOnt Port Types proposes a hierarchy of port types (interfaces), providing different granularity levels of access to the data model components for the sake of usability.

The upper levels of the hierarchy are general purpose interfaces that are fixed in the WS-DAIOnt specification and are mandatory for every underlying realization. The lower levels of the hierarchy are realization-dependent. In order to create the port types in a standard way, WS-DAIOnt defines a set of message design and organization criteria based on the components of the WS-DAIOnt data model usage and relationships patterns.

5. Conclusions

Ontology access provisioning is crucial if we want to enrich the Grid with semantic technologies. Furthermore, due to the increasing number of existing ontology languages and tools, an effective mechanism that guarantees interoperability between ontology access mechanisms must be developed. Up to date no protocols nor mechanisms are available in the OGSA architecture for dealing with ontologies in an effective manner.

By extending WS-DAI with WS-DAIOnt and the accompanying realizations, we provide the current grid architecture with a standard way of supplying ontology access and management capabilities, making ontologies available in grid environments like other specialized data resources usable across virtual organizations, thus enabling the future integration of semantic technologies in the grid architecture.

WS-DAIOnt, and the accompanying realizations, are still under development as part of the OntoGrid project.

Acknowledgments

We would like to thank all of those who have helped us anyhow: Sean Bechhofer, Óscar Corcho, Rosario Plaza and Mª del Carmen Suárez de Figueroa.

This work is supported by the OntoGrid project (FP6-511513) and by a U.P.M. pre-doctoral grant.

References

[1] M. Antonioletti, M. Atkinson, A. Krause, S. Malaika, S. Laws, N. W. Paton D. Pearson, and G. Riccardi. Web Services Data Access and Integration – The Core (WS-DAI) Specification, Version 1.0. GWD-R, Global Grid Forum, DAIS Working Group, Jun 2006.

[2] M. Antonioletti, B. Collins, A. Krause, S. Laws, J. Magowan, S. Malaika, and N.W. Paton. Web Services Data Access and Integration - The Relational Realisation (WS-DAIR) Specification, Version 1.0. GWD-R, Global Grid Forum, DAIS Working Group, Jun 2006.

[3] M. Antonioletti, S. Hastings, A. Krause, S. Langella, S. Laws, S. Malaika, and N.W. Paton. Web Services Data Access and Integration – The XML Realization (WS-DAIX)

Specification, Version 1.0. GWD-R, Global Grid Forum, DAIS Working Group, Jun 2006.

[4] S. Bechhofer. The DIG Description Logic Interface: DIG/1.1. Specification, DL Implementation Group (DIG), Feb 2003.

[5] J. Broekstra, A. Kampman, and F. van Harmelen. Sesame: A Generic Architecture for Storing and Querying RDF and RDF Schema. In I. Horrocks and J. Hendler, editors, *The Semantic Web - ISWC 2002: First International Semantic Web Conference*, number 2342 in Lecture Notes in Computer Science, pages 54–68. Springer, May 2002.

[6] B. Collins. Web Services Data Access and Integration - The File Realization (WS-DAIF). Informational recommendation, Global Grid Forum, DAIS Working Group, Oct 2004.

[7] D. De Roure, N. R. Jennings, and N. R. Shadbolt. The Semantic Grid: Past, Present and Future. *Proceedings of the IEEE*, 93(3):669–681, Mar 2005.

[8] I. Foster (Ed), D. Berry, A. Djaoui, A. Grimshaw, H. Kishimoto (Ed) B. Horn, F. Maciel, A. Savva (Ed), F. Siebenlist, R. Subramaniam, J. Treadwell, and J. Von Reich. The Open Grid Services Architecture, Version 1.5. GWD-I, Global Grid Forum, OGSA Working Group, Mar 2006.

[9] M. Esteban Gutiérrez (Ed), S. Bechhofer, O. Corcho, M. Fernández-López, A. Gómez-Pérez, Z. Kaoudi, I. Kotsiopulos, M. Koubarakis, M^{a} C. Suárez-Figueroa, and V. Tamma. Specification and Design of Ontology Grid Compliant and Grid Aware Services. Deliverable D3.1, OntoGrid Consortium, Apr 2005.

[10] Asunción Gómez-Pérez, Óscar Corcho, and Mariano Fernández-López. *Ontological Engineering : with examples from the areas of Knowledge Management, e-Commerce and the Semantic Web*. Advanced Information and Knowledge Processing. Springer, first edition, Jul 2004.

[11] V. Haarslev and R. Moeller. Racer: A core inference engine for the Semantic Web. In D. Fensel, K. P. Sycara, and J. Mylopoulos, editors, *The Semantic Web - ISWC 2003, Second International Semantic Web Conference*, number 2870 in Lecture Notes in Computer Science. Springer, Oct 2003.

[12] G. Karvounarakis, S. Alexaki, V. Christophides, D. Plexousakis, and M. Scholl. RQL: A Declarative Query Language for RDF. In *11th International World Wide Web Conference*. ACM, May 2002.

[13] E. Prud'hommeaux and A. Seaborne. SPARQL Query Language for RDF. Working draft, W3C, Feb 2006.

[14] A. Seaborne. RDQL – A Query Language for RDF. Member Submission, W3C, Jan 2004.

[15] Rudi Studer, V. Richard Benjamins, and Dieter Fensel. Knowledge engineering: Principles and methods. *Data & Knowledge Engineering*, 25(1–2):161–197, Mar 1998.